controls.js

This module encapsulates code responsible for managing local media and room config. This includes creation of new tracks, stopping existing tracks and preparation of room config before connecting to server.

1.Wrapper function

createControls() code

Wrapper function which keeps controls inside the closure

```
const createControls = function(config) {
```

2.Create controls object

code

Create controls object which will hold all the html handles. Also initialize tables for displaying local tracks configuration

```
const controls = {
    entrance: {
        url: document.getElementById("url"),
        roomName: document.getElementById("roomName"),
        roomPin: document.getElementById("roomPin"),
        nickName: document.getElementById("nickName"),
        enter: document.getElementById("startButton")
    },
    addVideoTrack: {
        source: document.getElementById("addVideoTrackSource"),
        width: document.getElementById("addVideoTrackWidth"),
        height: document.getElementById("addVideoTrackHeight"),
        codec: document.getElementById("addVideoTrackCodec")
    },
    addAudioTrack: {
        source: document.getElementById("addAudioTrackSource"),
        channels: document.getElementById("addAudioTrackChannels")
    },
    addVideoEncoding: {
        rid: document.getElementById("addVideoTrackEncodingRid"),
        active: document.getElementById("addVideoTrackEncodingActive"),
        maxBitrate: document.getElementById("addVideoTrackEncodingMaxBitrate"),
        resolutionScale: document.getElementById("addVideoTrackEncodingResolutionScale")
    },
    tables: {
        video: $('#videoTracksTable').DataTable({
            "sDom": 't',
            "columns": [
                {
                    "className":      'details-control',
                    "orderable":      false,
                    "data":           null,
                    "defaultContent": ''
                },
                {"data": "source"},
                {"data": "width"},
                {"data": "height"},
                {"data": "codec"},
                {"data": "action"}
            ]
        }),
        audio: $('#audioTracksTable').DataTable({
            "sDom": 't',
            "columns": [
                {"data": "source"},
                {"data": "channels"},
                {"data": "action"}
            ]
        }),
        encodings: $('#videoTrackEncodingsTable').DataTable({
            "sDom": 't',
            "columns": [
                {"data": "rid"},
                {"data": "active"},
                {"data": "maxBitrate"},
                {"data": "resolutionScale"},
                {"data": "action"}
            ]
        })
    }
}
```

3.Fill entrance modal window fields

code

Populate entrance modal with provided config

```
//apply room config
controls.entrance.url.value = config.room.url;
controls.entrance.roomName.value = config.room.name;
controls.entrance.roomPin.value = config.room.pin;
controls.entrance.nickName.value = config.room.nickName;
```

4.Add new audio tracks to controls

addAudioTrackRow() code

Function that will add new audio tracks to controls and notify main script

```
    const addAudioTrackRow = async function(track) {
        const stream = await getMedia([track]);
        let button = '<button id="' + stream.id + '-button" class="btn btn-primary">Delete</button>';
        const row = controls.tables.audio.row.add({
            source: track.source,
            channels: track.channels,
            action: button,
            stream: stream
        }).node();
        controls.tables.audio.draw();

        $('#' + stream.id + "-button").on('click', function(){
            //terminate stream
            console.log("terminate audio stream " + stream.id);
            let track = stream.getAudioTracks()[0];
            track.stop();
            track.dispatchEvent(new Event("ended"));
        }).prop('disabled', true);
        stream.getTracks()[0].onended = function() {
            controls.tables.audio.row(row).remove().draw();
        }
        trackCallback({
            stream: stream,
            encodings: track.encodings,
            source: track.source,
            type: track.type
        });
    }
```

Request local media from the WebRTC API

code

```
const stream = await getMedia([track]);
```

Add track to the audio tracks table

code

```
        let button = '<button id="' + stream.id + '-button" class="btn btn-primary">Delete</button>';
        const row = controls.tables.audio.row.add({
            source: track.source,
            channels: track.channels,
            action: button,
            stream: stream
        }).node();
        controls.tables.audio.draw();
```

Subscribe to "click" event. Once "Delete" button is clicked stop the track and dispatch "ended" event

code

```
        $('#' + stream.id + "-button").on('click', function(){
            //terminate stream
            console.log("terminate audio stream " + stream.id);
            let track = stream.getAudioTracks()[0];
            track.stop();
            track.dispatchEvent(new Event("ended"));
        }).prop('disabled', true);
```

Subscribe to track's "ended" event and clean the table once track is ended

code

```
stream.getTracks()[0].onended = function() {
    controls.tables.audio.row(row).remove().draw();
}
```

Notify main script we have a new local track

code

```
        trackCallback({
            stream: stream,
            encodings: track.encodings,
            source: track.source,
            type: track.type
        });
```

5.Add new video tracks to controls

addVideoTrackRow() code

Function that will add new video tracks to controls and notify main script. This is the same as addAudioTrackRow function except it adds video

```
    const addVideoTrackRow = async function(track) {
        const stream = await getMedia([track]);
        let button = '<button id="' + stream.id + '-button" class="btn btn-primary">Delete</button>';
        const row = controls.tables.video.row.add({
            source: track.source,
            width: track.width,
            height: track.height,
            codec: track.codec,
            action: button,
            stream: stream,
            encodings: track.encodings,
        }).node();
        controls.tables.video.draw();

        $('#' + stream.id + "-button").on('click', function(){
            //terminate stream
            console.log("terminate video stream " + stream.id);
            let track = stream.getVideoTracks()[0];
            track.stop();
            track.dispatchEvent(new Event("ended"));
        }).prop('disabled', true);
        stream.getTracks()[0].addEventListener("ended", function() {
            controls.tables.video.row(row).remove().draw();
        });
        trackCallback({
            stream: stream,
            encodings: track.encodings,
            source: track.source
        });
    }
```

6.Format video encodings

format() code

Helper function to format video encodings so we can display them in the table nicely

```
const format = function(d) {
    if (!d.encodings) {
        return;
    }
    let details =  '<table cellpadding="5" cellspacing="0" border="0" style="padding-left:50px;">';
    d.encodings.forEach(function(encoding){
        details += '<tr>';
        for (const [key, value] of Object.entries(encoding)) {
            details += '<td>'+ key + '</td>'+
                '<td>'+ value + '</td>';
        }
        details += '</tr>';
    });
    details +='</table>';
    return details;
}
```

7. Audio and video track tables displaying

displayTables() code

```
    const displayTables = async function() {
        // Add event listener for opening and closing details
        $('#videoTracksTableBody').on('click', 'td.details-control', function () {
            let tr = $(this).closest('tr');
            let row = controls.tables.video.row(tr);
            if (row.child.isShown()) {
                // This row is already open - close it
                row.child.hide();
                tr.removeClass('shown');
            } else {
                // Open this row
                row.child(format(row.data())).show();
                tr.addClass('shown');
            }
        });

        // Add preconfigured audio and video tracks
        for (const track of config.media.audio.tracks) {
            await addAudioTrackRow(track);
        }
        for (const track of config.media.video.tracks) {
            await addVideoTrackRow(track);
        }

        // Click event listener to add a new video track
        document.getElementById("addVideoTrack").addEventListener("click", function(e){
            let encodings = [];
            controls.tables.encodings.rows().every(function() {
                let encoding = this.data();
                encodings.push({
                    rid: encoding.rid,
                    active: encoding.active,
                    maxBitrate: encoding.maxBitrate,
                    scaleResolutionDownBy: encoding.resolutionScale
                })
            });
            let track = {
                source: controls.addVideoTrack.source.value,
                width: controls.addVideoTrack.width.value,
                height: controls.addVideoTrack.height.value,
```

```
            codec: controls.addVideoTrack.codec.value,
            encodings: encodings
        }
        addVideoTrackRow(track);
    });

    // Click event listener to remove video quality
    $("#videoTrackEncodingsTable").on("click", ".remove", function(){
        controls.tables.encodings.row($(this).parents('tr')).remove().draw();
    });

    // Click event listener to add video quality
    document.getElementById("addVideoTrackEncoding").addEventListener("click", function(){
        let button = '<button class="btn btn-primary remove">Delete</button>';
        controls.tables.encodings.row.add({
            rid: controls.addVideoEncoding.rid.value,
            active: controls.addVideoEncoding.active.value,
            maxBitrate: controls.addVideoEncoding.maxBitrate.value,
            resolutionScale: controls.addVideoEncoding.resolutionScale.value,
            action: button
        }).draw();
    });

    // Click event listener to add a new audio track
    document.getElementById("addAudioTrack").addEventListener("click", function(e){
        let encodings = [];
        let track = {
            source: controls.addAudioTrack.source.value,
            channels: controls.addAudioTrack.channels.value,
            encodings: encodings
        }
        addAudioTrackRow(track);
    });

}
```

7.1.Add handler to display/hide track details

code

Add open and close handler for video tracks details

```
    $('#videoTracksTableBody').on('click', 'td.details-control', function () {
        let tr = $(this).closest('tr');
        let row = controls.tables.video.row(tr);
        if (row.child.isShown()) {
            // This row is already open - close it
            row.child.hide();
            tr.removeClass('shown');
        } else {
            // Open this row
            row.child(format(row.data())).show();
            tr.addClass('shown');
        }
    });
```

7.2. Add audio and video tracks from configuration file to table

code

Add all configured audio and video tracks to the table

```
        // Add preconfigured audio and video tracks
        for (const track of config.media.audio.tracks) {
            await addAudioTrackRow(track);
        }
        for (const track of config.media.video.tracks) {
            await addVideoTrackRow(track);
        }
```

7.3.Add video tracks to the table

code

Add all configured video tracks to the table

```
        // Click event listener to add a new video track
        document.getElementById("addVideoTrack").addEventListener("click", function(e){
            let encodings = [];
            controls.tables.encodings.rows().every(function() {
                let encoding = this.data();
                encodings.push({
                    rid: encoding.rid,
                    active: encoding.active,
                    maxBitrate: encoding.maxBitrate,
                    scaleResolutionDownBy: encoding.resolutionScale
                })
            });
            let track = {
                source: controls.addVideoTrack.source.value,
                width: controls.addVideoTrack.width.value,
                height: controls.addVideoTrack.height.value,
                codec: controls.addVideoTrack.codec.value,
                encodings: encodings
            }
            addVideoTrackRow(track);
        });
```

7.4.Add audio tracks to the table

code

Add all configured audio tracks to the table

```
        // Click event listener to add a new audio track
        document.getElementById("addAudioTrack").addEventListener("click", function(e){
            let encodings = [];
            let track = {
                source: controls.addAudioTrack.source.value,
                channels: controls.addAudioTrack.channels.value,
                encodings: encodings
            }
            addAudioTrackRow(track);
        });
```

8.Mute form elements

muteForm() code

Define helper function for muting forms

```
const muteForm = function(form) {
    for (const [key, value] of Object.entries(form)) {
        value.disabled = true;
    }
}
```

9.Unmute form elements

unmuteForm() code

Define helper function for unmuting forms

```
const unmuteForm = function(form) {
    for (const [key, value] of Object.entries(form)) {
        value.disabled = false;
    }
}
```

10.Mute entrance modal window inputs

muteInput() code

Define function that mutes entrance inputs

```
const muteInput = function() {
    muteForm(controls.entrance);
}
```

11.Create room configuration object

roomConfig() code

Define function that will assemble room config

```
    const roomConfig = function() {
        let roomConfig = {
            url: controls.entrance.url.value,
            roomName: controls.entrance.roomName.value,
            pin: controls.entrance.roomPin.value,
            nickname: controls.entrance.nickName.value
        };
        if (config.room.failedProbesThreshold !== undefined) {
            roomConfig.failedProbesThreshold = config.room.failedProbesThreshold;
        }
        if (config.room.pingInterval !== undefined) {
            roomConfig.pingInterval = config.room.pingInterval;
        }
        return roomConfig;
    }
```

12.Get local video tracks

getVideoStreams() code

Define function that will return all available local video tracks

```
    const getVideoStreams = function() {
        let streams = [];
        controls.tables.video.rows().every(function(rowIdx, tableLoop, rowLoop) {
            let data = this.data();
            streams.push({
                stream: data.stream,
                encodings: data.encodings,
                source: data.source,
                type: data.type
            });
        });
        return streams;
    }
```

13.Get local audio tracks

getAudioStreams() code

Define function that will return all available local audio tracks

```
    const getAudioStreams = function() {
        let streams = [];
        controls.tables.audio.rows().every(function(rowIdx, tableLoop, rowLoop) {
            let data = this.data();
            streams.push({
                stream: data.stream,
                encodings: [],
                source: data.source
            });
        });
        return streams;
    }
```

14.Handler to add video track to the table

code

The function passes a callback function to a new tracks

```
const onTrack = function(callback) {
    trackCallback = callback;
}
```

15.Export functions

code

Export functions for the script

```
    return {
        muteInput: muteInput,
        roomConfig: roomConfig,
        displayTables: displayTables,
        getAudioStreams: getAudioStreams,
        getVideoStreams: getVideoStreams,
        onTrack: onTrack,
        cleanTables: cleanTables
    }
```

16.Get media streams from WebRTC API

getMedia() code

Requests local media streams from the WebRTC API

```
const getMedia = async function(tracks) {
    //convert to constraints
    let screen = false;
    const constraints= {};
    tracks.forEach(function(track){
        if (track.source === "mic") {
            //audio
            constraints.audio = {};
            if (track.constraints) {
                constraints.audio = track.constraints;
            }
            constraints.audio.stereo = track.channels !== 1
            if (track.channels && track.channels === 2) {
                constraints.audio.echoCancellation = false;
                constraints.audio.googEchoCancellation = false;
            }
        } else if (track.source === "camera") {
            constraints.video = {};
            if (track.constraints) {
                constraints.video = track.constraints;
            }
            constraints.video.width = track.width;
            constraints.video.height = track.height;
        } else if (track.source === "screen") {
            constraints.video = {};
            if (track.constraints) {
                constraints.video = track.constraints;
            }
            constraints.video.width = track.width;
            constraints.video.height = track.height;
            screen = true;
        }
    });

    //get access to a/v
    let stream;
    if (screen) {
        stream = await navigator.mediaDevices.getDisplayMedia(constraints);
    } else {
        stream = await navigator.mediaDevices.getUserMedia(constraints);
    }
    return stream;
}
```